Procedural Generation of Initial States of Sokoban

Dâmaris S. Bento,[1] **André G. Pereira**[2] and **Levi H. S. Lelis**[1]
[1]Universidade Federal de Viçosa, Brazil
[2]Universidade Federal do Rio Grande do Sul, Brazil
{damaris.bento, levi.lelis}@ufv.br, agpereira@inf.ufrgs.br

Abstract

Procedural generation of initial states of state-space search problems have applications in human and machine learning as well as in the evaluation of planning systems. In this paper we deal with the task of generating hard and solvable initial states of Sokoban puzzles. We propose hardness metrics based on pattern database heuristics and the use of novelty to improve the exploration of search methods in the task of generating initial states. We then present a system called β that uses our hardness metrics and novelty to generate initial states. Experiments show that β is able to generate initial states that are harder to solve by a specialized solver than initial states designed by human experts.

1 Introduction

The development of search solvers for state-space search problems is a traditional area of research in artificial intelligence (AI), where notable progress has been made (e.g., [Culberson and Schaeffer, 1996; Junghanns and Schaeffer, 2001; Hoffmann and Nebel, 2001; Holte *et al.*, 2016]). By contrast, systems for generating problems have achieved limited success. Generation systems are often able to generate only small and easy problems, thus limiting their applicability.

Systems able to automatically generate solvable state-space search problems have applications in human and machine learning. These systems can be used to generate problems from which humans [Anderson *et al.*, 1985] and systems [Arfaee *et al.*, 2011; Lelis *et al.*, 2016; Racanière *et al.*, 2017; Orseau *et al.*, 2018] can learn. Automatically generated problems are also used to evaluate planning systems. Problems used in the International Planning Competition are usually generated using domain knowledge and their solvability verified with specialized solvers. The generation of solvable and hard problems can be challenging, since it could be PSPACE-complete to decide if a problem is solvable. Moreover, it is unclear what makes a problem hard in practice.

In this paper we focus on the task of generating hard and solvable initial states of Sokoban. We introduce hardness metrics based on pattern databases (PDBs) [Culberson and Schaeffer, 1996] motivated by hardness metrics known to correlate with the time required by humans to solve problems [Jarušek and Pelánek, 2010]. We also propose the use of novelty [Lipovetzky and Geffner, 2012] to improve the exploration in the task of generating initial states. Finally, we introduce a system called β that performs a search guided by our metrics of hardness and novelty, and that also uses our metrics to select hard and solvable initial states.

Although β is in principle general and applicable to other problem domains, we focus on Sokoban for several reasons. First, Sokoban is a PSPACE-complete problem [Culberson, 1999] and a traditional testbed for search methods. Second, in Sokoban most of the states in its space are unsolvable, thus making the problem of generating solvable initial states particularly challenging. Third, Sokoban has a community of expert designers who create hard problems and make them available online, thus allowing a comparison between human-designed initial states with initial states generated by β.

Our empirical results show that β is able to generate solvable initial states that are harder to solve by a specialized solver than those designed by human experts. To the best of our knowledge, our work is the first to compare the hardness of initial states generated by a computer system with those designed by human experts.

Contributions We introduce computationally efficient hardness metrics and show empirically how these metrics can be used to guide a search algorithm and to select hard instances from of a pool of options. Another contribution is to show that novelty [Lipovetzky and Geffner, 2012] is essential for generating a diverse pool of solvable instances from which hard ones can be selected. The combination of our metrics and novelty resulted in the first generative system for Sokoban with superhuman performance in terms of instance difficulty for a specialized solver.

2 Related Work

Related to our work are papers presenting methods for generating entire problems of Sokoban, and more generally, works dealing with automatic content generation. Also related to our work are learning systems that require the generation of a set of states from which one can learn a value or a heuristic function. Next, we briefly survey each of these related areas.

Murase *et al.* [1996], Taylor and Parberry [2011], and Kartal *et al.* [2016] presented methods for generating complete

Sokoban problems, where a complete problem is composed of a maze and an initial state. Although the problem of generating complete problems is interesting, we focus on the task of generating initial states and assume that the maze is provided as input. This way we can directly compare the initial states β generates with those designed by human experts on a fixed set of mazes. Further, β can be used to generate initial states for the problems generated by previous systems.

The systems of Murase *et al.* [1996] and Taylor and Parberry [2011] have been applied to generate a set of problems from which reinforcement learning (RL) algorithms learn [Racanière *et al.*, 2017]. The drawback of using these systems is that they are unable to generate large problems, thus limiting RL algorithms to small and easy problems. Since our method generates hard initial states for large Sokoban problems, it will allow RL algorithms to be evaluated in more challenging problems. In the same line of research, Justesen *et al.* [2018] showed how systems that generate video game levels can enhance deep RL algorithms.

Arfaee *et al.* [2011] describe a system that generates a set of initial states of a given problem and learns a heuristic function while solving this set of states with IDA* [Korf, 1985]. Arfaee *et al.* experimented with domains for which it is easy to generate a set of solvable states with varied difficulty (e.g., sliding-tile and pancake puzzles). Before our work, Arfaee *et al.*'s system could not be applied to problem domains in which it is hard to generate difficult and solvable states, such as Sokoban. The combination of Arfaee *et al.*'s system with β is a promising direction of future research.

Procedural Content Generation (PCG) methods are used to generate content such as game levels [Snodgrass and Ontañón, 2017], puzzles [Smith and Mateas, 2011; Sturtevant and Ota, 2018], and educational problems [Polozov *et al.*, 2015]. Many of the problem domains used in the PCG literature (e.g., math problems for children) are computationally easier than the PSPACE-complete domain we consider in this paper. Moreover, in contrast with our work, PCG systems are often concerned with properties such as the enjoyability of the generated problems [Mariño and Lelis, 2016], as opposed to the scalability of the system. Finally, to the best of our knowledge, β is the first PCG method employing concepts such as PDBs and novelty to the task of problem generation.

3 Background and Problem Definition

Although we focus on the generation of initial states of Sokoban puzzles, our system is general and we describe it as such. Let $\mathcal{P} = \langle S, s_0, S^*, A\rangle$ be a *state-space search problem*. A state is a complete assignment over a set of variables $\mathcal{V}$, where each variable $v \in \mathcal{V}$ has a finite set domain D_v. Let S be a set of states, $s_0 \in S$ an initial state, $S^* \subseteq S$ a set of goal states, A a finite set of actions. An action $a \in A$ is a pair $\langle pre_a, post_a\rangle$ specifying the preconditions and postconditions (effect) of action a. pre_a and $post_a$ are assignments of values to a subset of variables in $\mathcal{V}$, denoted $\mathcal{V}_{pre}$ and $\mathcal{V}_{post}$. Action a is applicable to state s, denoted $s[\![a]\!]$, if s agrees with $\mathcal{V}_{pre}$. The result of $s[\![a]\!]$ is state s' that agrees with $post_a$ for the variables in $\mathcal{V}_{post}$ and with s for the remaining variables. We say that the application of a to s *generates* state s'.

We refer to the states that can be generated from state s as *successor states*, denoted $\mathrm{succ}(s)$. We assume that for each action $a \in A$ there is an *inverse action* $a^{-1} \in A^{-1}$ such that $s' = a[\![s]\!]$ iff $s = a^{-1}[\![s']\!]$. Here, A^{-1} is the set of inverse actions that can applied to s, which yield a set of *predecessor states*, denoted $\mathrm{pred}(s)$. A state s is *expanded* when either $\mathrm{succ}(s)$ or $\mathrm{pred}(s)$ is invoked.

A *solution path* is a sequence of actions that transforms s_0 in a state $s_* \in S^*$. An *optimal solution path* is a solution path with minimum length. $h(s)$ is a heuristic function that estimates the length to reach a goal state from s, with h^* providing the optimal solution length for all states $s \in S$. A heuristic h is *admissible* if $h(s) \leq h^*(s)$ for all $s \in S$.

We can now define the task we are interested in solving.

Definition 1 (initial state generation)**.** *An initial state generation task is defined as $\mathcal{P}_{-s_0} = \langle S, S^*, A\rangle$. $\mathcal{P}_{-s_0}$ is a state-space search problem $\mathcal{P}$ without a initial state s_0. In problem $\mathcal{P}_{-s_0}$ one has to provide a state $s \in S$ such that P_{-s_0} with s as initial state is solvable.*

4 Pattern Databases

A *pattern database* (PDB) is a memory-based heuristic function defined by a *pattern* V for a problem $\mathcal{P}$ [Culberson and Schaeffer, 1996; Edelkamp, 2001]. A pattern is a subset of variables $V \in \mathcal{V}$ that induces an abstracted state-space of $\mathcal{P}$. The abstracted space is obtained by ignoring the information of variables $v \notin V$. For a given pattern, the optimal solutions of all states in the induced abstracted space is computed and stored in a look-up table. These stored values can then be used as a heuristic function estimating h^*. The combination of multiple PDB heuristics can result in better estimates of h^* than the estimates provided by each PDB alone. PDBs can be combined by taking the maximum [Holte *et al.*, 2006] or the sum [Felner *et al.*, 2004] of multiple PDB estimates. The heuristic function resulting from the sum of multiple PDBs is called an additive PDB heuristic. An additive PDB is admissible if no action affects more than one pattern V, where an action a affects V if a has an effect on any of its variables, i.e., $\mathcal{V}_{post_a} \cap V \neq \emptyset$. Consider $\mathcal{P}_e$, the search problem shown in Figure 1, and the example that follows.

$$
\begin{array}{rcl}
\mathcal{V} & = & \{v_1, v_2, v_3\},\\
D_{v_i} & = & \{0,1,2,3,4\} \text{ for all } v_i \in \mathcal{V},\\
s_0 & = & \{v_1 = 0, v_2 = 0, v_3 = 0\},\\
s_* & = & \{v_1 = 3, v_2 = 3, v_3 = 3\},\\
A & = & \{inc_x^v | v \in \mathcal{V}, x \in \{0,1,2\}\} \cup \{jump^v | v \in \mathcal{V}\},\\
 & & inc_x^{v_i} = \langle v_i = x, v_i = x+1\rangle,\\
 & & jump_{v_i} = \langle \mathcal{V} - v_i = 4 \wedge v_i = 0, v_i = 3\rangle,
\end{array}
$$

Figure 1: Example adapted from Pommerening *et al.* [2013].

$\mathcal{P}_e$ has three variables that can be assigned a value in $\{0,1,2,3,4\}$. The initial state assigns 0 to all variables, and the goal state has all variables with the value of 3. $\mathcal{P}_e$ has two types of actions $inc_x^{v_i}$ and $jump_{v_i}$. Actions $inc_x^{v_i}$ change the value of the variable v_i from x to $x+1$. Actions $jump_{v_i}$ change the value of variable v_i from 0 to 3, as long as all the other variables have the value of 4. All solutions to this problem involve the use of action $inc_x^{v_i}$ three times to all three

variables, yielding an optimal solution length of 9. $jump_{v_i}$ cannot be used in any solution because if a variable has the value of 4, the problem becomes unsolvable since there is no action to change its value.

Example 1. *Let $\{\{v_1\},\{v_2,v_3\}\}$ be a collection with two additive patterns for $\mathcal{P}_e$. The heuristic estimate returned for $\mathcal{P}_e$'s s_0 with a PDB heuristic with pattern $\{v_1\}$ is 1, as a single action $jump$ changes the value of v_1 from 1 to 3, thus achieving the goal in the PDB's abstracted space. The heuristic estimate returned for s_0 for pattern $\{v_2,v_3\}$ is 6. This is because the action $jump$ is no longer applicable. Since the two patterns are additive, an additive PDB with $\{\{v_1\},\{v_2,v_3\}\}$ returns $1+6=7$ as an estimate of the solution cost of s_0.*

5 Proposed Search Algorithm for β

Given a problem $\mathcal{P}_{-s_0}$, β performs a backward greedy best-first search [Doran and Michie, 1966] from the set S^*. This is performed by inserting all states $s_* \in S^*$ in a priority queue denoted `open` and in a set of generated states denoted `closed`. β expands states from `open` according to an order $[\cdot]$, which β receives as input. When a state s is removed from `open`, β expands s invoking $\mathrm{pred}(s)$. Every $s' \in \mathrm{pred}(s)$ that is not in `closed` is inserted in `open` and `closed`.

The advantage of performing a backward search is that all states generated that way are guaranteed to have a solution. By contrast, if one assigned random values from the variables' domains as a means of generating initial states, then one would need to prove the resulting $\mathcal{P}$ problem to be solvable, and in some domains, this task is computationally hard. β returns the state s encountered in search with largest *objective value* once a time or a memory limit is reached. We specify below the objective functions used with β.

5.1 Solution Length

We propose two metrics of difficulty: solution length and conflicts. The optimal solution length is traditionally used to evaluate the hardness of initial states for AI methods. For example, solution length correlates with the running time of heuristic search algorithms in problems such as Rubik's Cube [Lelis *et al.*, 2013]. We use PDB heuristics to approximate the optimal solution length of state-space problems.

5.2 Conflicts

Intuitively, the hardness of state-space problems arises from the interactions among variables. If sub-problems defined by single variables of the problem can be solved independently, then the problem tends to be easy. However, if one needs to consider interactions between all variables of the problem, then the problem tends to be hard. We use PDBs to capture the interaction of variables in a problem. Let h^{PDB_k} be a PDB heuristic with patterns of size k. The *conflicts* of state s is defined as follows.

Definition 2 (conflicts)**.** *The number n of* conflicts *of order k, denoted kC, for $k>1$ of a state s is $n = h^{PDB_k}(s) - h^{PDB_{k-1}}(s)$, if $n>0$; $n=0$, otherwise.*

Suppose we have the following PDB heuristics for our running example: h^{PDB1}, h^{PDB2} and h^{PDB3}. h^{PDB1} uses three additive patterns $\{\{v_1\},\{v_2\},\{v_3\}\}$, which results in $h^{\mathrm{PDB1}}(s_0)=3$ (the action $jump$ is applicable to each pattern). h^{PDB2} uses two additive patterns, one with two variables and another with one variable: $\{\{v_1\},\{v_2,v_3,\}\}$. For these patterns $h^{\mathrm{PDB2}}(s_0)=7$. Finally, h^{PDB3} uses one pattern with all three variables and returns the optimal solution of 9.

Using these PDB-values we can compute the conflicts of s_0. s_0 has four conflicts of order two, denoted $2C$ ($h^{\mathrm{PDB2}}(s_0) - h^{\mathrm{PDB1}}(s_0) = 4$) and two conflicts of order three, denoted $3C$ ($h^{\mathrm{PDB3}}(s_0) - h^{\mathrm{PDB2}}(s_0) = 2$). The number of $2C$ conflicts approximates the effort in terms of solution length due to the interaction of pairs of variables. In general, the number of kC conflicts approximates the effort due to the interaction of k variables.

5.3 Novelty

Next, we discuss novelty, which we use to enhance the exploration of our search algorithm. Novelty w is a structural measure proposed to enhance the exploration of search algorithms in the task of finding solutions for search problems [Lipovetzky and Geffner, 2012; Lipovetzky and Geffner, 2017]. Novelty evaluates states with respect to the order in which an algorithm generates them.

Definition 3 (novelty)**.** *Let $\mathcal{P}$ be a state-space search problem, $s \in S$ be a state generated by a search algorithm, and h be a heuristic function. The* novelty *of s for h is $|\mathcal{V}| - n + 1$ if s is the first generated state during search with heuristic value $h(s)$ and with a subset of n variables being assigned values $d_{v_1},\cdots,d_{v_n}$ and no smaller subset of variables has this property.*

The novelty of a state s generated during the search is maximal if, for the first time in search, s has a variable v for which the value $d_v \in D_v$ is assigned. The novelty of s is minimum if $n = |\mathcal{V}|$, which means that it requires all variables in $\mathcal{V}$ to differ s from previously generated states. Intuitively, a state has larger novelty if it requires a smaller number of variables to assume a previously unseen assignment of values.[1]

Example 2. *Suppose we run β in $\mathcal{P}_e$ with the ordering $[\cdot]$ defined by novelty and that all states have the same heuristic value. State s_* has novelty $w(s_*)=3$ because this is the first state to assign 3 to v_1. Then, β invokes $\mathrm{pred}(s_*)$ and generates three states s_1,s_2,s_3 by applying the inc^{-1} (the inverse of inc) actions to each variable. For simplicity, we will assume that $jump^{-1}$ is not applicable. s_1,s_2,s_3 have $w=3$ because each of them assigns 2 to a variable for the first time. Next, suppose β expands $s_1=\{v_1=2,v_2=3,v_3=3\}$, then it generates three states $\{v_1=1,v_2=3,v_3=3\}$, $\{v_1=2,v_2=2,v_3=3\}$ and $\{v_1=2,v_2=3,v_3=2\}$, where the last two states have $w=2$ because the smallest subset with newly assigned values is two: $\{v_1=2,v_2=2\}$ for the second and $\{v_1=2,v_3=2\}$ for the third.*

[1] Our semantics for novelty differs from Lipovetzky and Geffner's semantics because they dealt with minimization problems while we deal with a maximization problem.

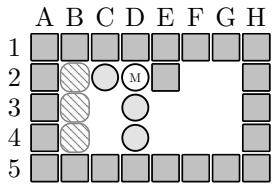


Figure 2: Sokoban problem.

We hypothesize that novelty can improve the exploration of the search performed by β, thus allowing it to visit a diverse set of states from which one can select initial states that maximize solution length and number of conflicts.

6 Difficulty Metrics for Humans in Sokoban

Jarušek and Pelánek [2010] performed an extensive user study showing strong positive correlations between metrics of difficulty they introduced with the time required by humans to solve Sokoban problems. In this section we introduce the domain of Sokoban and present their metrics.

6.1 Sokoban

Sokoban is a transportation problem played in a maze grid. The maze is defined by locations occupied by immovable blocks (walls), free (inner) locations and goal (inner) locations. An initial state of Sokoban is a placement of k boxes and a man in inner locations of the maze. The goal is to push, through the man, all boxes from their initial location to a goal location while minimizing the number of pushes.

Figure 2 shows a Sokoban maze with a wall at A1, an free location at C3 and goal location at B4. An initial state is defined by the assignment of the locations of boxes and the man. A state has a $k + 1$ variables, one for each box and another for the man. The instance has k goal locations, one for each box. The domain of each variable is the set of inner locations. Goal states have all k boxes at one of the k goal locations. Figure 2 shows a box at C2 and the man at D2. We name goal locations and boxes by their locations in the maze. For example, the box at C2 is box C2. An action corresponds to the man pushing a box to an adjacent empty inner location. The precondition of an action is that the man must be able to reach the orthogonal location adjacent to the box. For example, in Figure 2 box C2 can be pushed to B2, but box D3 cannot be pushed anywhere.

6.2 Jarušek and Pelánek's Metrics

The first metric introduced by Jarušek and Pelánek [2010] was solution length, which is exactly what we approximate with PDBs. Jarušek and Pelánek showed that the exact solution length yields a Spearman correlation of 0.47 with the time required by humans to solve Sokoban problems.

Jarušek and Pelánek [2010] introduced a metric named "number of counterintuitive moves". In Figure 2, while it is intuitive to push box C2 to a goal location (push it to the left), the pushes required to move boxes D3 and D4 to a goal location are "counterintuitive". This is because one needs to push either D3 or D4 away from a goal location before pushing them to a goal location. These counterintuitive pushes can be approximated by our metric of conflicts. In Figure 2, a PDB that considers only one box estimates that one can directly push all three boxes to the goal, yielding an estimate of $1 + 2 + 2 = 5$ pushes: one for C2 and two for D3 and D4. A PDB that considers two boxes captures the interaction between D3 and D4. Such a PDB discovers that one needs to push either D3 or D4 two locations to the right, before pushing them to a goal location, for an estimated solution length of $1 + 6 + 2 = 9$. The difference $9 - 5 = 4$ is the number of Jarušek and Pelánek's counterintuitive moves captured by our metric of conflicts. Jarušek and Pelánek showed that the exact number of conflicts has a Spearman correlation of 0.69 with the time required by humans to solve Sokoban problems.

Jarušek and Pelánek [2010] also introduced "problem decomposition", a metric that measures the degree in which subsets of boxes can be independently pushed to the goal. This metric yields a Spearman correlation of 0.82 with the time required by humans to solve the puzzles. Our order k of conflicts approximates similar property. If k is the largest value for which $h^{\text{PDB}_k}(s) - h^{\text{PDB}_{k-1}}(s) > 0$ for all possible PDBs with k and $k - 1$ variables, then the largest number of variables with counterintuitive moves in s is k. Although one might be unable to divide s into independent subproblems with k variables (e.g., the order in which the subproblems are solved might matter), one is able to reason about the subproblems with at most k variables somewhat independently.

The metrics introduced by Jarušek and Pelánek [2010] are computationally hard in general as they require one to solve Sokoban problems to compute their values. By contrast, our metrics can be computed efficiently and thus be used within search procedures for generating hard initial states.

7 Empirical Evaluation

We use the 90 problems of the xSokoban benchmark in our experiments. These problems were developed by human experts and are known to be challenging to both humans and AI methods. The xSokoban benchmark allows us to directly compare the initial states our method generates with those created by human experts in terms of the metrics proposed and number of problems solved by a solver. Moreover, the $\mathcal{P}_{-s_0}$ problems present in xSokoban contain many more boxes (the largest problem has 34 boxes) than the problems generated by the current generative methods (the largest problem reported in the literature has 7 boxes).

We instantiate variants of β by varying the ordering $[\cdot]$ in which the states are expanded and how initial states are selected. $[\cdot]$ is composed by a list of features, which can include novelty $w(h)$, a PDB heuristic h^{PDBk}, and the number of conflicts of order k, denoted kC. β returns the state generated with largest ordering value, ignoring novelty, once the search reaches the time or memory limit. For example, β with ordering $[w(h^{\text{PDB4}}), 4C, h^{\text{PDB4}}]$ expands states with largest $w(h^{\text{PDB4}})$ first, with the ties broken according to 4C and then according to h^{PDB4}. β returns the state s with largest 4C-value, with ties broken according to h^{PDB4}; remaining ties are broken by the state generation order.

Heuristic $h^{\text{PDBk'}}$ returns the maximum heuristic value for a set with $|\mathcal{V}| + 1$ additive PDBs. We use this number of PDBs

	#	Generation Method	States Expanded	h-value of s_0				Number of Conflicts			# Solved
				h^{PDB1}	h^{PDB2}	h^{PDB3}	h^{PDB4}	2C	3C	4C	
A	1	$[h^{\text{PDB1}}]$	36,844,914.09	179.68	185.94	187.88	190.49	6.27	1.93	2.61	33.0 ± 0.0
	2	$[h^{\text{PDB2}}]$	35,746,724.37	178.70	190.33	190.45	193.35	11.63	0.11	2.90	35.0 ± 0.0
	3	$[h^{\text{PDB3}}]$	16,762,063.67	166.28	175.87	178.09	180.63	9.58	2.22	2.54	33.8 ± 1.8
	4	$[h^{\text{PDB4}}]$	19,683,953.17	168.01	177.78	179.26	183.63	9.77	1.48	4.37	32.4 ± 2.8
B	5	$[h^{\text{PDB2}}, 2C]$	35,279,521.51	179.08	191.31	191.32	194.31	12.23	0.01	2.99	36.0 ± 0.0
	6	$[h^{\text{PDB3}}, 3C]$	14,896,276.53	167.41	176.91	179.70	182.31	9.49	2.79	2.62	33.2 ± 1.9
	7	$[h^{\text{PDB4}}, 4C]$	10,127,736.49	157.09	166.24	167.84	172.41	9.15	1.60	4.57	33.0 ± 2.5
C	8	$[2C, h^{\text{PDB2}}]$	23,465,651.94	136.89	158.97	154.48	158.78	22.08	0	4.30	26.0 ± 0.0
	9	$[3C, h^{\text{PDB3}}]$	10,075,985.22	111.44	117.76	126.85	127.69	6.31	9.10	0.83	28.8 ± 0.8
	10	$[4C, h^{\text{PDB4}}]$	6,066,057.87	101.95	111.48	112.42	123.57	9.54	0.94	11.15	31.0 ± 4.9
D	11	$[w(h^{\text{PDB1}}), h^{\text{PDB1}}]$	24,852,925.84	**223.43**	231.11	233.79	237.11	7.68	2.68	3.32	26.6 ± 0.5
	12	$[w(h^{\text{PDB2}}), h^{\text{PDB2}}]$	23,764,256.42	219.52	**233.19**	**234.14**	**238.07**	13.67	0.96	3.92	29.8 ± 0.4
	13	$[w(h^{\text{PDB3}}), h^{\text{PDB3}}]$	12,320,342.90	215.49	226.89	230.16	233.44	11.40	3.27	3.28	25.4 ± 1.1
	14	$[w(h^{\text{PDB4}}), h^{\text{PDB4}}]$	12,643,046.69	190.76	202.03	204.12	210.70	11.27	2.08	6.59	25.8 ± 1.5
E	15	$[w(h^{\text{PDB2}}), h^{\text{PDB2}}, 2C]$	26,994,420.86	216.29	230.81	231.22	235.19	14.52	0.40	3.97	27.0 ± 0.0
	16	$[w(h^{\text{PDB3}}), h^{\text{PDB3}}, 3C]$	11,842,771.67	211.28	222.30	225.45	228.89	11.02	3.15	3.45	26.0 ± 1.2
	17	$[w(h^{\text{PDB4}}), h^{\text{PDB4}}, 4C]$	7,650,277.40	207.38	218.77	221.24	227.21	11.39	2.47	5.97	26.2 ± 2.9
F	18	$[w(h^{\text{PDB2}}), 2C, h^{\text{PDB2}}]$	15,280,827.99	175.46	201.34	196.69	202.62	**25.89**	0	5.93	22.8 ± 0.4
	19	$[w(h^{\text{PDB3}}), 3C, h^{\text{PDB3}}]$	7,412,556.34	154.76	162.73	175.78	175.85	7.96	**13.06**	0.07	20.6 ± 2.1
	20	$[w(h^{\text{PDB4}}), 4C, h^{\text{PDB4}}]$	3,986,192.80	148.37	159.96	161.09	176.33	11.60	1.14	15.24	19.5 ± 2.4
	21	Aggregation	79,699,027.84	151.46	163.07	161.78	187.00	11.63	0.77	**25.22**	$\mathbf{16.4 \pm 1.3}$
-		Humans (xSokoban)	-	188.02	199.58	205.62	211.41	11.56	6.04	5.79	18

Table 1: Results of states generated by variants of β. The "h-value of s_0" denotes the average heuristic value of the generated initial states; "Number of Conflicts" shows the average number of conflicts, where 2C, 3C, and 4C denote the number of conflicts of order 2, 3, and 4, respectively. "# Solved" denotes the average and standard deviation of the number of problems solved by PRB.

to ensure that instances with more variables have more PDBs. The patterns of each of the $|\mathcal{V}| + 1$ additive PDBs are defined by iteratively and randomly selecting without replacement k' variables from $\mathcal{V}$, until all variables are in a pattern. If $|\mathcal{V}| \mod k' > 0$, then the last pattern will have $|\mathcal{V}| \mod k'$ variables. All patterns include the man.

The solver we use to evaluate the hardness of the instances generated by β and by human experts is the one introduced by Pereira *et al.* [2015], which we refer to as PRB.

We are primarily interested in comparing the states generated β with those designed by human experts in terms of problems solved by PRB. We also present the average number of states expanded by each variant of β while generating the 90 initial states, the average h-value of the initial states generated in terms of h^{PDB1}, h^{PDB2}, h^{PDB3} and h^{PDB4}, and the average number of conflicts 2C, 3C, and 4C. We also experimented with random walks (RW) and breadth-first search (BFS) as the search algorithms used by β and PRB solves almost all problems generated by these approaches (approximately 85 out of 90). This is because both methods generate initial states with very short solution length. RW and BFS results are omitted from the table of results for clarity.

All experiments are run on 2.66 GHz CPUs, β is allowed 1 hour of computation time and 8 GB of memory, while the solver is allowed 10 minutes and 4 GB (more memory and running time have a small impact on PRB). Due to the randomness of the PDBs, we report the average results of 5 runs of each approach. We also report the standard deviation for the number of problems PRB solves. Table 1 presents the results of our proposed method. The third column of the table ("Generation Method") shows the order used to guide the β search and to select initial states in our experiments. We also present the number of states β expanded, number of problems solved by PRB, and the h-values and conflict values, the table also presents a column "#" indicating the row of each method, which are used as a reference in our discussion. We highlight in bold of cells with the best result for a given criterion. For example, $[w(h^{\text{PDB1}}), h^{\text{PDB1}}]$ is the best performing method in terms of value of h^{PDB1}.

7.1 Quantitative Results

We start by comparing the β results without novelty (see blocks A, B and C in Table 1) with those with novelty (blocks D, E and F). Methods that use novelty outperform their counterparts in all criteria: average h-value, number of conflicts, and the total number of problems solved. Compare, for example, the results in rows 1–4 with their counterparts in rows 11–14. In particular, $[w(h^{\text{PDB1}}), h^{\text{PDB1}}]$ obtained an average h^{PDB1}-value of 223.43, which is larger than the value of 188.02 for the average of the xSokoban initial states. Although $[w(h^{\text{PDB1}}), h^{\text{PDB1}}]$ can generate initial states with larger h^{PDB1} values than the xSokoban, its initial states

tend to be much easier in terms of the number of problems solved than the xSokoban. PRB solves at least seven more initial states generated by the methods in block D than the initial states created by human experts.

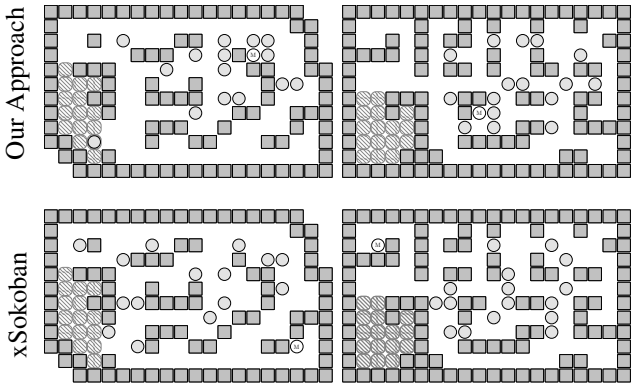
The methods shown in blocks B and E maximize the heuristic value and break ties by conflict values. Methods in blocks C and F use the opposite order of metrics. The methods in B generate initial states with longer solution lengths than the methods in C. However, the methods in C have a larger number of conflicts, according to the optimized criteria. For example, the method in row 8, which prioritizes the maximization of 2C-values, generates initial states with larger 2C-values than its counterpart in row 5. In general, the initial states generated by the methods in C tend to be much harder in terms of the number of problems solved by PRB. These results demonstrate the importance of generating initial states with a large number of conflicts. The same pattern is observed by comparing the results of the methods in E with those of the methods in F. The results in F highlight the importance of combining the maximization of conflicts of higher order with novelty to enhance exploration. The method shown in line 20 is already competitive with the xSokoban benchmark in terms of number of problems solved by PRB.

The method presented in line 21 shows the average numbers of 5 runs of an aggregation procedure. One run of this procedure represents 20 runs of $[w(h^{\text{PDB4}}), 4C, h^{\text{PDB4}}]$. Then, out of the 20 runs, we select for each maze the initial state that maximizes $[4C, h^{\text{PDB4}}]$. This procedure further optimizes the average 4C-value (25.22) at the cost of a larger number of state expansions and it outperforms the initial states designed by humans in terms of number of problems PRB is able to solve. PRB solves 18 xSokoban instances and only an average of 16.4 instances of the aggregation procedure.

The metric that correlated to most with the time required by humans to solve Sokoban problems in Jarušek and Pelánek's study was "problem decomposition," which is similar to our order of conflict. Our results suggest that states with many conflicts of high order tend to be harder to solve by PRB. Taken together with Jarušek and Pelánek's user study, our results suggest that some of the properties that make problems hard for humans can also make them hard for AI systems.

7.2 Qualitative Results

Figure 3 shows representative initial states for two xSokoban mazes. The initial states shown at the top were generated by a representative run of $[w(h^{\text{PDB4}}), 4C, h^{\text{PDB4}}]$ and at the bottom are the original xSokoban problems. These initial states are visually similar since there are no obvious features that distinguish those states generated by β from those created by humans. The states generated by our method have larger values of h^{PDB4} for these mazes: 219 and 223, while the values are 167 and 202 for xSokoban. PRB solves the xSokoban problem for maze #53, but does not solve the β state for the same maze. Although both initial states have the same 4C-value for maze #53 ($= 2$), our initial state has a much longer solution length, which could explain the result. The situation is reversed for maze #62, where PRB solves the initial state generated by our approach, but does not solve the initial state designed by humans, and both states also have the same 4C-value ($= 6$). We conjecture that the xSokoban state for maze #62 has conflicts of order higher than 4, which our PDBs are unable to capture. These conflicts of higher order could make the xSokoban problem harder to solve than the one generated by our system.



(a) #53. (b) #62.

Figure 3: Initial states of Sokoban generated by our best variant of β (top row), and the initial states of xSokoban (bottom row).

Currently, no solver is able to solve all xSokoban problems. Yet, humans can solve all of them. Thus, humans are still superior to AI methods in the task of solving Sokoban problems. By contrast, our results show that β achieves superhuman performance in the task of generating initial states in terms of instance difficulty for a solver.

Although we evaluated β only on Sokoban, we believe it can generalize to other domains. This is because all techniques used in our system are Sokoban independent. Moreover, Sokoban is a prototypical problem of a large class of problems known as transportation problems. Problems in this class has four central characteristics: there is a set of connected locations, a set of movable objects and a set of goal locations, and the solution is a sequence of actions that move (a subset of) the movable objects to (a subset of) the goal locations. Sokoban presents all these characteristics thus our system is likely to perform well in these problems as well.

8 Conclusions

In this paper we presented a system for generating hard and solvable initial states of Sokoban. Our system β performs a backward search from the goal states of a problem and selects states using PDB-based hardness metrics. We also proposed the use of novelty to guide β's search. Compared to the states designed by human experts, β is able to generate states that are harder to solve by a solver. β is the first generative system of initial states of Sokoban with superhuman performance in terms of difficulty for a solver.

Acknowledgments

André Pereira acknowledges support from FAPERGS with project $17/2551-0000867-7$ and Levi Lelis acknowledges support from FAPEMIG and CNPq. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

[Anderson *et al.*, 1985] John R Anderson, C Franklin Boyle, and Brian J Reiser. Intelligent tutoring systems. *Science*, 228:456–462, 1985.

[Arfaee *et al.*, 2011] Shahab Jabbari Arfaee, Sandra Zilles, and Robert Craig Holte. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175:2075–2098, 2011.

[Culberson and Schaeffer, 1996] Joseph C. Culberson and Jonathan Schaeffer. Searching with pattern databases. In *Canadian Conference on Artificial Intelligence*, pages 402–416, 1996.

[Culberson, 1999] Joseph C. Culberson. Sokoban is PSPACE-Complete. In *Fun With Algorithms*, pages 65–76, 1999.

[Doran and Michie, 1966] Jim E Doran and Donald Michie. Experiments with the graph traverser program. In *Royal Society of London A*, volume 294, pages 235–259. The Royal Society, 1966.

[Edelkamp, 2001] Stefan Edelkamp. Planning with pattern databases. In *European Conference on Planning*, pages 13–24, 2001.

[Felner *et al.*, 2004] Ariel Felner, Richard E. Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22:279–318, 2004.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14(1):253–302, 2001.

[Holte *et al.*, 2006] Robert C. Holte, Ariel Felner, Jack Newton, Ram Meshulam, and David Furcy. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence*, 170(16–17):1123–1136, 2006.

[Holte *et al.*, 2016] Robert C Holte, Ariel Felner, Guni Sharon, and Nathan R Sturtevant. Bidirectional search that is guaranteed to meet in the middle. In *AAAI Conference on Artificial Intelligence*, volume 16, pages 3411–3417, 2016.

[Jarušek and Pelánek, 2010] Petr Jarušek and Radek Pelánek. Difficulty rating of sokoban puzzle. In *Starting AI Researcher Symposium*, pages 140–150, 2010.

[Junghanns and Schaeffer, 2001] Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129:219–251, 2001.

[Justesen *et al.*, 2018] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Procedural level generation improves generality of deep reinforcement learning. *CoRR*, abs/1806.10729, 2018.

[Kartal *et al.*, 2016] Bilal Kartal, Nick Sohre, and Stephen J Guy. Data-driven sokoban puzzle generation with monte carlo tree search. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2016.

[Korf, 1985] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree earch. *Artificial Intelligence*, 27(1):97–109, 1985.

[Lelis *et al.*, 2013] Levi H. S. Lelis, Sandra Zilles, and Robert C. Holte. Predicting the Size of IDA*'s Search Tree. *Artificial Intelligence*, pages 53–76, 2013.

[Lelis *et al.*, 2016] Levi H. S. Lelis, Roni Tzvi Stern, Shahab Jabbari Arfaee, Sandra Zilles, Ariel Felner, and Robert Craig Holte. Predicting optimal solution costs with bidirectional stratified sampling in regular search spaces. *Artificial Intelligence*, 230:51–73, 2016.

[Lipovetzky and Geffner, 2012] Nir Lipovetzky and Héctor Geffner. Width and serialization of classical planning problems. In *European Conference on Artificial Intelligence*, pages 540–545. IOS Press, 2012.

[Lipovetzky and Geffner, 2017] Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In *AAAI Conference on Artificial Intelligence*, pages 3590–3596, 2017.

[Mariño and Lelis, 2016] Julian R. H Mariño and Levi H. S. Lelis. A computational model based on symmetry for generating visually pleasing maps of platform games. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 65–71, 2016.

[Murase *et al.*, 1996] Yoshio Murase, Hitoshi Matsubara, and Yuzuru Hiraga. Automatic making of sokoban problems. *PRICAI: Topics in Artificial Intelligence*, pages 592–600, 1996.

[Orseau *et al.*, 2018] Laurent Orseau, Levi H. S. Lelis, Tor Lattimore, and Théophane Weber. Single-agent policy tree search with guarantees. In *Advances in Neural Information Processing Systems*, pages 3205–3215, 2018.

[Pereira *et al.*, 2015] André G Pereira, Marcus Ritt, and Luciana S Buriol. Optimal sokoban solving using pattern databases with specific domain knowledge. *Artificial Intelligence*, 227:52–70, 2015.

[Polozov *et al.*, 2015] Oleksandr Polozov, Eleanor O'Rourke, Adam M. Smith, Luke Zettlemoyer, Sumit Gulwani, and Zoran Popovic. Personalized mathematical word problem generation. In *International Joint Conference on Artificial Intelligence*, pages 381–388. AAAI, 2015.

[Pommerening *et al.*, 2013] Florian Pommerening, Gabriele Röger, and Malte Helmert. Getting the most out of pattern databases for classical planning. In *International Joint Conference on Artificial Intelligence*, 2013.

[Racanière *et al.*, 2017] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5690–5701, 2017.

[Smith and Mateas, 2011] Adam M. Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):187–200, 2011.

[Snodgrass and Ontañón, 2017] Sam Snodgrass and Santiago Ontañón. Player movement models for video game level generation. In *International Joint Conference on Artificial Intelligence*, pages 757–763, 2017.

[Sturtevant and Ota, 2018] Nathan R. Sturtevant and Matheus Jun Ota. Exhaustive and semi-exhaustive procedural content generation. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 109–115, 2018.

[Taylor and Parberry, 2011] Joshua Taylor and Ian Parberry. Procedural generation of sokoban levels. In *International North American Conference on Intelligent Games and Simulation*, pages 5–12, 2011.